

FIG. 2

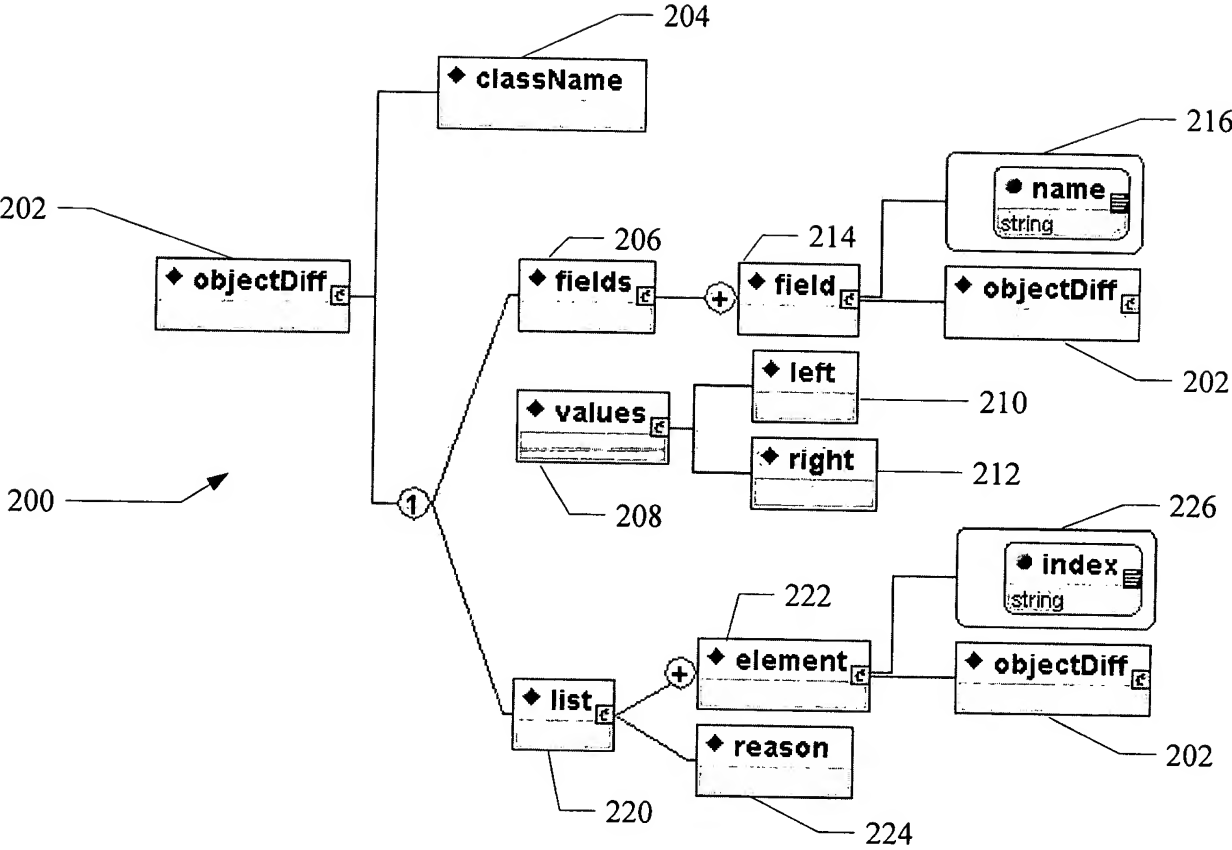
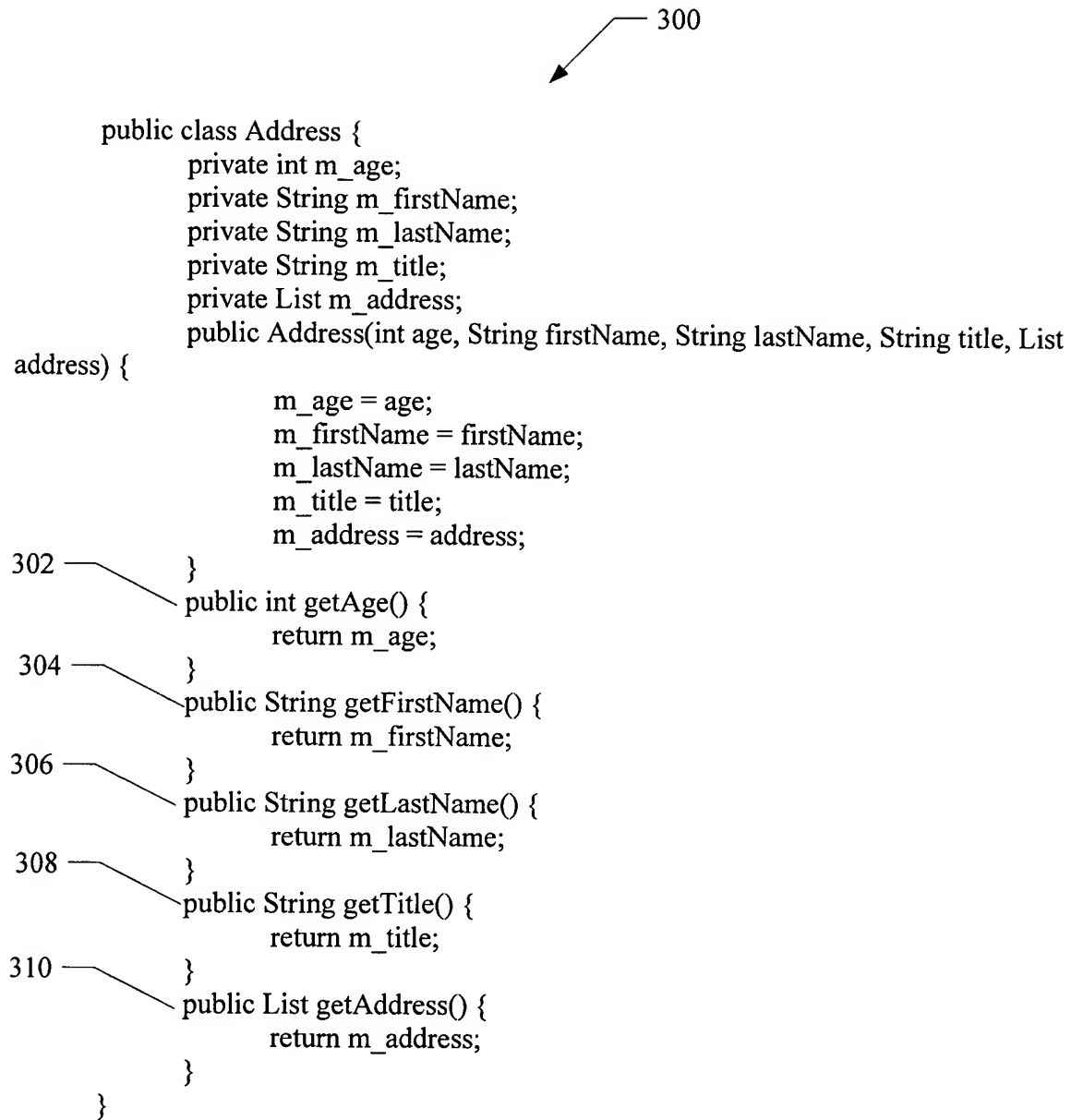


FIG. 3



```

public class Address {
    private int m_age;
    private String m_firstName;
    private String m_lastName;
    private String m_title;
    private List m_address;
    public Address(int age, String firstName, String lastName, String title, List
address) {
        m_age = age;
        m_firstName = firstName;
        m_lastName = lastName;
        m_title = title;
        m_address = address;
    }
    public int getAge() {
        return m_age;
    }
    public String getFirstName() {
        return m_firstName;
    }
    public String getLastName() {
        return m_lastName;
    }
    public String getTitle() {
        return m_title;
    }
    public List getAddress() {
        return m_address;
    }
}

```

300

302

304

306

308

310

FIG. 4

400 —————>

```
public void testAddressDiff() {  
402 —————>    List a1 = new ArrayList();  
                  a1.add("1 Underhole");  
                  a1.add("Cartoon Land");  
                  Address addr1 = new Address(13, "Bugs", "Bunny", "Mr", a1);  
  
404 —————>    List a2 = new ArrayList();  
                  a2.add("2 Underhole");  
                  a2.add("Cartoon Land");  
                  Address addr2 = new Address(11, "Barb", "Bunny", "Ms", a2);  
  
                  Document d = ObjectDiff.compare( addr1, addr2 );  
                  DocumentHelper.write(d, System.out);  
}
```

```

<?xml version="1.0" encoding="UTF- 8"?>
<objectDiff>
  <className>com.chase.gmdr.base.xml.test.ObjectDiffTest$Address</className>
  <fields>
    <field name="age" different="true">
      <objectDiff>
        <className>java.lang.Integer</className>
        <toString>
          <left>13</left>
          <right>11</right>
        </toString>
      </objectDiff>
    </field>
    <field name="firstName" different="true">
      <objectDiff>
        <className>java.lang.String</className>
        <toString>
          <left>Bugs</left>
          <right>Barb</right>
        </toString>
      </objectDiff>
    </field>
    <field name="lastName" different="false">
      <toString>Bunny</toString>
    </field>
    <field name="title" different="true">
      <objectDiff>
        <className>java.lang.String</className>
        <toString>
          <left>Mr</left>
          <right>Ms</right>
        </toString>
      </objectDiff>
    </field>
    <field name="address" different="true">
      <objectDiff>
        <className>java.util.ArrayList</className>
        <list>
          <element index="0">
            <objectDiff>
              <className>java.lang.String</className>
              <toString>
                <left>1 Underhole</left>
                <right>2 Underhole</right>
              </toString>
            </objectDiff>
          </element>
        </list>
      </objectDiff>
    </field>
  </fields>
</objectDiff>

```

← 500

FIG. 6

```

/*****
ObjectDiff.java

Copyright (2000) Chase Manhattan Corporation
All rights reserved
*****/

package com.chase.gmdr.base.xml;

import java.util.*;
import java.lang.reflect.*;
import org.w3c.dom.*;
import com.chase.gmdr.base.service.logging.*;

/**
 * Utility methods for comparing two objects.
 *
 * Copyright (2000) Chase Manhattan Corporation All rights reserved
 *
 * @author John Phenix
 */
public class ObjectDiff
{
    /**
     * Interface to define a list of methods that should be ignored by the getMethods method
     * @see com.chase.gmdr.base.xml.ObjectDiff#getMethods
     */
    public interface MethodsToIgnore
    {
        public List getMethodsToIgnore();
    }

    /**
     * Some static constants that specify the reason that two object differ using the compareObjects method.
     * @see com.chase.gmdr.base.xml.ObjectDiff#compareObjects
     */
    public interface CompareReasons
    {
        public final static int ONE_IS_NULL = 100;
        public final static int DIFFERENT_CLASSES = 101;
        public final static int DIFFERENT_LIST_SIZES = 102;
        public final static int UNABLE_TO_COMPARE = 103;
    }

    /**
     * Creates a DOM Document describing the differences between two objects.
     */

```

FIG. 7

```

public final static Document compare( Object a, Object b )
{
    Document doc = null;

    try
    {
        doc = DocumentHelper.newDocument();
    }

    catch (java.io.IOException e)
    {
        FailureManager.log(ObjectDiff.class, "compare", "Unable to create new Document ", e);
        Errors.log(ObjectDiff.class, "compare", "Unable to create new Document", e);
    }

    Node n = createObjectDiffNode(doc, a, b);
    if (n != null)
        doc.appendChild(n);
    else
        doc = null;

    return doc;
}

/**
 * Compare two objects.
 * Can return a CompareReasons constant.
 * If the object is a List then compareLists is called.
 * If the object is a Comparable then compareTo is called, otherwise equals is called.
 * @return 0 if the objects are equal.
 */
public final static int compareObjects(Object a, Object b)
{
    if (a == null && b == null)
        return 0;
    if ((a == null && b != null) || (a != null && b == null))
        return CompareReasons.ONE_IS_NULL;
    if (!a.getClass().equals(b.getClass()))
        return CompareReasons.DIFFERENT_CLASSES;
    if (a instanceof List)
        return compareLists((List)a, (List)b);
    else if (a instanceof Comparable)
        return ((Comparable)a).compareTo(b);
    else
        return a.equals(b) ? 0 : 1;
}

```

FIG. 8

```
/**
 * Compares the results of the get methods (as returned from the getMethods method)
 * from the two objects.
 * Can return a CompareReasons constant.
 * @return 0 if the results from the get methods of the two objects are equal.
 */
public static int compareGetMethods(Object a, Object b)
{
    if (a == null && b == null)
        return 0;
    if ((a == null && b != null) || (a != null && b == null))
        return CompareReasons.ONE_IS_NULL;
    if (!a.getClass().equals(b.getClass()))
        return CompareReasons.DIFFERENT_CLASSES;

    Method[] methods = getMethods(a);
    for (int i=0; i<methods.length; i++)
    {
        Object resa=null, resb=null;

        try
        {
            resa = methods[i].invoke( a, null );
            resb = methods[i].invoke( b, null );
        }

        catch ( Exception e )
        {
            FailureManager.log(ObjectDiff.class, "compareGetMethods", "unable to compare ", e);
            return CompareReasons.UNABLE_TO_COMPARE;
        }

        int result = compareObjects(resa, resb);
        if (result != 0)
            return result;
    }
    return 0;
}

/**
 * Compare two lists.
 * Can return a CompareReasons constant.
 * Calls compareObjects on each element of the list.
 * @return 0 if the objects are equal.
 */
public final static int compareLists(List a, List b)
{
```


FIG. 9

```
        if (a == null && b == null)
            return 0;
        if ((a == null && b != null) || (a != null && b == null))
            return CompareReasons.ONE_IS_NULL;
        if (a.size() != b.size())
            return CompareReasons.DIFFERENT_LIST_SIZES;
        if (a.size() == 0)
            return 0;
        int size = a.size();
        for (int i=0; i<size; i++)
        {
            Object oa = a.get(i);
            Object ob = b.get(i);

            int c = compareObjects(oa, ob);
            if (c != 0)
                return c;
        }
        return 0;
    }

    private static Map s_methodMap = new HashMap();
    /**
     * Returns a list of methods for the specified object.
     * The methods must start with "get" and have no arguments.
     * The method getClass is not returned.
     * If the object implements GetMethodsToIgnore then these methods are ignored as well.
     * @return An array of Method
     */
    public final static Method[] getMethods(Object a)
    {
        Class c = a.getClass();
        /**
         * Cache the methods for each Class
         */
        if (s_methodMap.containsKey(c))
            return (Method[])s_methodMap.get(c);

        Method[] methods = c.getMethods();
        List newMethods = new ArrayList();

        List methodsToIgnore = null;
        if (a instanceof MethodsToIgnore)
            methodsToIgnore = ((MethodsToIgnore)a).getMethodsToIgnore();

        for( int i=0; i<methods.length; i++ )
        {
```

```

// if method begins with get and has no parameters
String methodName = methods[i].getName();
if( methodName.substring( 0, 3 ).equals("get")
&& methods[i].getParameterTypes().length == 0)
{
    if(methodsToIgnore != null && (methodsToIgnore.contains(methodName)||methodName.equals("getThis"))
        continue;
    if(methodName.equals("getClass")||methodName.equals("getThis"))
        continue;
    newMethods.add(methods[i]);
}
}

int size = newMethods.size();
Method[] nMethods = new Method[size];
for (int i=0; i<size; i++)
    nMethods[i] = (Method)newMethods.get(i);

s_methodMap.put(c, nMethods);
return nMethods;
}

/**
 * Create an objectDiff node that shows the difference between two objects.
 */
private static Node createObjectDiffNode(Document doc, Object oa, Object ob )
{
    // Compare oa and ob
    int c = compareObjects(oa, ob);
    switch (c)
    {
        case CompareReasons.DIFFERENT_CLASSES:
        {
            if (!(oa instanceof List))
            {
                Node n = doc.createElement("differentClass");
                n.appendChild(createValues(doc, oa.getClass().getName(), ob.getClass().getName())
                    return n;
            }
            break;
        }
        case CompareReasons.ONE_IS_NULL:
        {
            Node n = doc.createElement( "objectDiff" );
            n.appendChild( createElement(doc, "ClassName", oa != null ? oa.getClass().getName():ob.g
                // Values
                n.appendChild(createValues(doc, oa,ob));
            return n;
        }
    }
}

```

FIG. 11

ATTY DOCKET No.: 14846-21

```

    }
    case 0:
        return null;
    }

    if (Trace.ON)
        Trace.log(ObjectDiff.class, "createObjectDiffNode", "DIFF class="+oa.getClass().getName());

    Node n = doc.createElement( "ObjectDiff" );
    n.appendChild( createElement(doc, "className", oa.getClass().getName() ) );

    if (oa instanceof List)
        createListNode(doc, n, (List)oa, (List)ob);
    else if (oa.getClass() == String.class || oa instanceof java.util.Date || oa instanceof Number)
        n.appendChild(createValues(doc, oa,ob));
    else
        createFieldsNode(doc, n, oa, ob);

    return n;
}
/**
 * Create list node.
 * Shows an element node for each element that is different in the two lists
 */
private static void createListNode(Document doc, Node n, List oa, List ob)
{
    // List
    Node list = doc.createElement("list");
    n.appendChild(list);

    int size = oa.size();
    if (Trace.ON)
        Trace.log(ObjectDiff.class, "createObjectDiffNode", "List size="+size);
    if (size != ob.size())
    {
        Node reason = doc.createElement("listsAreDifferentSizes");
        reason.appendChild( createElement(doc, "left", Integer.toString(oa.size()) ) );
        reason.appendChild( createElement(doc, "right", Integer.toString(ob.size()) ) );
        list.appendChild(reason);
    }
    else
    {
        for (int i=0; i<size; i++)
        {
            // Now for each field create an ObjectDiff for each field
            Node elementDiff = createObjectDiffNode(doc, oa.get(i),ob.get(i));

```

FIG. 12

```

        if (elementDiff != null)
        {
            Node element = doc.createElement("element");
            ((Element)element).setAttribute("index", Integer.toString(i));
            element.appendChild(elementDiff);
            list.appendChild(element);
        }
    }

    /**
     * Creates a fields node, that lists each field that has a get method.
     * If the field is the same on both objects the value is simply output,
     * if the field is different an objectDiff node is created to show the differences.
     */
    private static void createFieldsNode(Document doc, Node n, Object oa, Object ob)
    {
        Node fields = doc.createElement("fields");
        n.appendChild(fields);

        Method[] methods = getMethods(oa);
        for (int i=0; i<methods.length; i++)
        {
            String methodName = methods[i].getName();

            Object resa=null, resb=null;
            try {
                resa = methods[i].invoke( oa, null );
                resb = methods[i].invoke( ob, null );
            }

            catch ( Exception e )
            {
                FailureManager.log(ObjectDiff.class, "createFieldsNode", "Unable to compare ", e);
            }

            boolean different = compareObjects(resa, resb) != 0;

            Node field = doc.createElement("field");
            ((Element)field).setAttribute("name", fieldName(methodName));
            if (different)
            {
                ((Element)field).setAttribute("different", "true");
                field.appendChild(createObjectDiffNode(doc, resa, resb));
            }
            else
            {

```

FIG. 13

ATTY DOCKET No.: 14846-21

```

        ((Element)field).setAttribute("different", "false");
        field.appendChild(createElement(doc, "toString", resa == null ? "null" : resa.toString())
    )
        fields.appendChild(field);
    }

    /**
     * Creates element
     */
    private static Node createElement(Document doc, String type, String value )
    {
        Node n = doc.createElement( type );
        n.appendChild( doc.createTextNode( value ) );
        return n;
    }

    /**
     * Create a toString node
     */
    private static Node createValues(Document doc, Object oa, Object ob)
    {
        // Values
        Node values = doc.createElement( "toString" );
        values.appendChild( createElement(doc, "left", oa == null ? "null" : oa.toString() ) );
        values.appendChild( createElement(doc, "right", ob == null ? "null" : ob.toString() ) );
        return values;
    }

    private static Map s_fieldNames = new HashMap();
    /**
     * Change a method name to a field name; e.g. getFred to fred.
     * Cache in a HashMap because this will get called far many more times than there are method names.
     */
    private static String fieldName(String methodName)
    {
        if (s_fieldNames.containsKey(methodName))
            return (String)s_fieldNames.get(methodName);
        StringBuffer sb = new StringBuffer(methodName);
        sb.delete(0,3);
        sb.setCharAt(0, Character.toLowerCase(sb.charAt(0)));
        String name = sb.toString();
        s_fieldNames.put(methodName, name);
        return name;
    }
}

```